VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Algoritmy pre výpočet optického toku s využitím GPU/CUDA

CUDA technology in the algorithms of computer graphics and digital image processing

2012 Ľuboš Suchár

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Ľuboš Suchár**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Algoritmy pro výpočet optického toku s využitím GPU/CUDA

CUDA Technology in the Algorithms of Computer Graphics and Digital Image Processing

Zásady pro vypracování:

Výpočet optického toku je jedním z důležitých problémů analýzy dynamických scén. Cílem diplomové práce je implementovat některý z osvědčených algoritmů v prostředí GPU/CUDA.

V bakalářské práci proveďte:
1. Seznamte se s problematikou výpočtu optického toku, algoritmy a aplikacemi.
2. Zvolte jeden z algoritmů, který naimplementujete v jazyce C/C++.
3. Algoritmus z předchozího kroku naimplementujte pro prostředí GPU/CUDA.
4. V textové části práce podrobně popište dosažené výsledky, zejména dosažené urychlení.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Dr. Ing. Eduard Sojka**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 30.4.2012

Šuchár .....

Na tomto mieste by som rád poďakoval pánovi doc. Dr. Ing. Eduardovi Sojkovi, ktorý ma vďaka plnohodnotným konzultáciam viedol k úspešnému zvládnutiu tejto práce.

Abstrakt

Výpočet optického toku je jedným z dôležitých problémov analýzy dynamických scén. Práca oboznamuje s problematikou výpočtu optického toku, algoritmami a aplikáciami. Bol zvolený jeden z algorimov, ktorý bol naimplementovaný v jazyku C/C++ a následne bol implementovaný pre prostredie GPU/CUDA. V práci je popísané urýchlenie, ktoré bolo dosiahnuté použitím tejto technológie.

Kľúčové slová: analýza obrazu, analýza pohybu, CUDA, optický tok, GPGPU

Abstract

Calculation of optical flow is one of important problems in analysis of dynamic scenes. This paper acquitants with issues connected with calculation of optical flow, algorithms and their aplications. One of these algorithms was choosen and implemented in language C/C++ and then implemented for CUDA environment. This paper describes the acceleration achieved with use of this technology.

Keywords: CUDA, image analysis, motion analysis, optical flow, GPGPU

Zoznam použitých skratiek a symbolov

2D	–	2 dimensional	–	Dvojrozmerný
3D	–	3 dimensional	–	Trojrozmerný
CPU	–	Central processing unit	–	Centrálna procesná jednotka
CUDA	–	Compute Unified Device Architecture	–	Zjednotená architekrúra pre počítanie
GPU	–	Graphics processing unit	–	Grafická procesná jednotka
GPGPU	–	General-purpose computing on graphics processing units	–	Univerzálne výpočty na grafickej procesnej jednotke
RAM	–	Random access memory	–	Pamät's náhodnym prístupom do pamäte
SAD	–	Sum of absolute differences	–	Suma absolútnych hodnôt rozdielov
SIMD	–	Single instruction, multiple data	–	Jedna inštrukcia, viacej dát

Seznam tabulek

Seznam obrázků

Obsah

1 Úvod

Zrak je náš najdôležitejší zmysel, ktorým vnímame až 80% podnetov z vonkajšieho prostredia. Pri pohľade na pohybujúcu sa scénu vie ľudský zrak priam v okamihu spracovať a vyhodnotiť túto situáciu. Vie jednoznačne identifikovať predmety, detekovať pohyb, určiť pohybujúci sa objekt a odhadnúť jeho nasledujúcu pozíciu. Pre počítač je táto úloha veľmi zložitá. Veda ktorá sa snaží túto schopnosť napodobiť v počítači sa nazýva počítačové videnie. Jednou z hlavných častí počítačového videnia je bezpochyby analýza pohybu. S analýzou pohybu sa v súčasnosti stretávame veľmi často, napríklad pri meraní rýchlosti automobilov, v bezpečnostných systémoch alebo aj v antikolíznych systémoch. Analýza pohybu umožňuje, aby tieto veci boli schopné vykonávať počítače automaticky bez zásahu človeka. Výpočty s tým spojené sú však veľmi vypočtovo a časovo náročné.

Spoločnosť nVidia, jeden z najznámejších výrobcov grafických kariet a iných počítačových súčastí, pri svojich výskumoch vynašla technológiu, ktorá pri zložitých výpočtoch môže poskytnúť až niekoľkonásobne lepší výkon. Táto technológia sa nazýva „Compute Unified Device Architecture“ a označuje sa skratkou CUDA. Umožňuje nám vykonávať vysoko paralelné aplikácie a tým výrazne znížiť čas vykonávania algoritmu. Preto by bolo určite zaujímavé porovnať rýchlosť vykonávania výpočtu spojeného s analýzou pohybu na klasickom procesore a na tejto grafickej karte.

Cieľom tejto práce je oboznámiť s problémom výpočtu optického toku a jeho algoritmami. Vybrať jeden z týchto algoritmov, naimplementovať v programovacom jazyku C/C++ a následne pre prostredie CUDA, vykonať testy a zmeriať urýchlenie, ktoré nám toto prostredie poskytne.

Bakalárska práca začína popisom problému a uvedením do problematiky analýzy pohybu. Druhá kapitola je venovaná výpočtu rozdielovej metódy a určeniu optického toku. V ďalšej kapitole sú popísané metódy určenia optického toku Lucas-Kanade, Horn-Schunck a Block matching, ktorá bola implementovaná a preto jej je venovaná väčšia časť. Piata kapitola je venovaná GPGPU a technológii CUDA. Kapitoly 6 a 7 sú venované implementácií algoritmu a jeho testovaním.

2 Analýza pohybu

Z hľadiska využitia v praxi môžeme uvažovať o troch hlavných typoch úloh analýzy pohybu. Prvým typom je detekcia pohybu na snímkach. Cieľom druhej skupiny úloh je nájsť polohu pohybujúcich sa objektov, prípadne objekty rozpoznať a opísať. Tretia skupina úloh, ktorá využíva informáciu o pohybe umožňuje určiť trojrozmerné vlastnosti objektov s využitím ich dvojrozmerných projekcií získaných v rôznych časových okamihoch pohybu.

2.1 Dynamický obraz

Dynamické obrazy sú v podstate postupnosti statických obrazov. Zo statického obrázka môžeme získať informáciu o polohe objektov v obraze, ale nezistíme nič o smere, rýchlosti alebo zrýchlení pohybu. Tento problém sa rieši postupnosťou statických obrázkov u ktorých je známy čas medzi zachytením jednotlivých snímkov.

Pri zriadovaní snímkov je niekoľko možností vzťahov medzi kamerou a objektom. Podľa toho sa potom volí vhodná metóda pre analýzu pohybu. Môžu nastať tieto možnosti:

- kamera v pokoji, objekty v pokoji,
- kamera v pohybe, objekty v pokoji,
- kamera v pokoji, objekty v pohybe,
- kamera v pohybe, objekty v pohybe.

Digitálny dynamický obraz zachytený fotoaparátom alebo kamerou môžeme vyjadriť funkciou $f(x, y, t)$, kde x a y sú súradnice bodu v čase t. Pri spracovaní obrazu sa pracuje s obrazovými funkciami, ktoré sú reprezentované maticami. Hodnotou tejto funkcie je jas. Čiže pre každý bod v tomto obraze poznáme len jednu informáciu.

Správnosť väčšiny algoritmov spojených s analýzou pohybu v obraze je založená na predpoklade nazývanom *brightness constancy contraint*, ktorý tvrdí, že zmeny jasu v dynamickom obraze sú spôsobené iba pohybom. Faktory iné ako pohyb, ktoré môžu spôsobiť zmenu jasu, ako napríklad zmeny v osvetlení scény zachytenej v obraze môžu byť ignorované za predpokladu, že interval medzi snímkami je dostatočne krátky. Z toho dôvodu je pre real-time využitie týchto výpočtov nevyhnutná vysoká rýchlosť snímkov.

2.2 Druhy pohybov

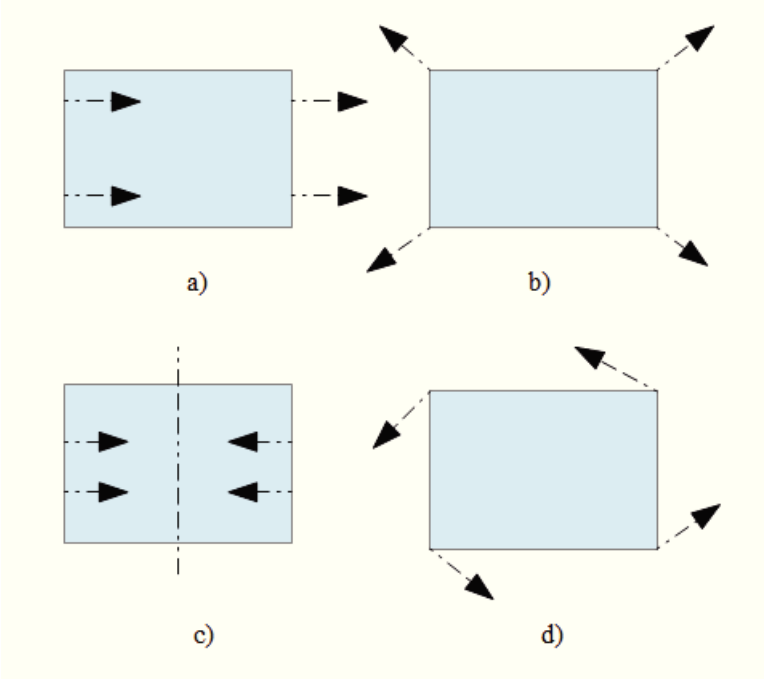
V dynamických obrazoch sa môže vyskytovať mnoho pohybov, avšak všetky pohyby môžeme popísať pomocou štyroch základných základných pohybov:

a) translačný pohyb v rovine kolmej na os pohľadu,

b) translačný pohyb do diaľky,

c) rotácia kolmá na os pohľadu,

d) rotácia okolo osi pohľadu.



Obrázok 1: Druhy pohybov

3 Existujúce riešenia problému

3.1 Rozdielová metóda

Pravdepodobne najjednoduchšou metódou na detekciu pohybu v obraze je rozdielová metóda [1]. Umožňuje nám zistiť rozdiely medzi snímanými obrazmi v rôznych časových okamžikoch stacionárnou kamerou v konštantnom osvetlení. Rozdielový obraz d, je taký binárny obraz, že hodnoty 0 predstavujú sebe odpovedajúce miesta obrazov f_1 a f_2, v ktorých nedošlo k významným zmenám jasu v čase medzi snímaním obrazov. Tento obraz môžeme teda zapísať ako

$$D_{1,2} = \begin{cases} 0 & pre \quad |f_1(i,j) - f_2(i,j)| < e \\ 1 & pre \quad |f_1(i,j) - f_2(i,j)| \geq e \end{cases}, \tag{3.1}$$

kde e je predom určené kladné číslo. Týmto zistíme, že body kde rozdielový obraz nadobúda hodnotu 0 sa v čase nezmenili a hodnoty, kde nadobúda hodnotu 1 sa v čase zmenili. Zmena v čase môže byť detekovaná z nasledujúcich príčin:

- obrazový element $f_1(i,j)$ bol elementom pohybujúceho sa objektu a $f_2(i,j)$ bol elementom nepohybujúceho sa pozadia, alebo naopak,
- obrazový element $f_1(i,j)$ bol elementom pohybujúceho a $f_2(i,j)$ elementom iného pohybujúceho sa objektu,
- obrazové elementy $f_1(i,j)$ a $f_2(i,j)$ boli elementy toho istého pohybujúceho sa objektu v miestach rôzneho jasu,
- vplyvom prítomnosti šumu a rôznych nepresností snímania statickou kamerou sa v rozdielovom obraze objaví množstvo nesprávne detekovaných elementov s hodnotou 1.

Tieto chyby metódy môžu byť jednoducho potlačené nastavením hodnoty e. Avšak príliš veľká hodnota môže spôsobiť, ze nebudú detekované pomalé pohyby a pohyby malých objektov.

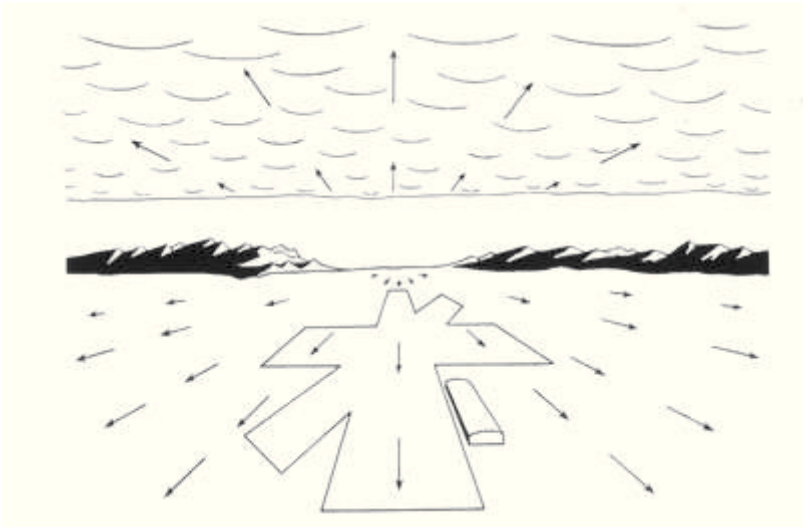
Z rozdielového obrazu 2 môžeme úspešne detekovať pohyb v obrazoch, ale nemôžeme určiť smer pohybu objektu. Na určenie smeru pohybu je vhodnejšia metóda optického toku. Ktorá je popísaná v ďalšej kapitole.

Obrázok 2: Použitie rozdielovej metódy na snímkoch z križovatky

3.2 Optický tok

Pojem optický tok odkazuje na vizuálne javy ktoré zažívame každý deň. Optický tok je v podstate jasný vizuálny pohyb, ku ktorému dochádza pri pohybe objektov. Predstavte si, že sedíme v aute, alebo vo vlaku a pozeráme von oknom. Vidíme stromy, budovy, cesty atď. Vyzerá to akoby sa pohybovali späť. Tento pohyb je optický tok. Tento pohyb nám môže tiež povedať ako blízko sme k rôznym objektom, ktoré vidíme. Vzdialené objekty ako oblaky, alebo hory sa pohybujú tak pomaly, že to vyzerá ako keby stáli na mieste. Objekty, ktoré sú bližšie ako napríklad budovy a stromy vyzerajú ako keby sa pohybovali smerom naspäť, pričom bližšie objekty sa pohybujú rýchlejšie ako vzdialené objekty.

Existujú jasné matematické vzťahy medzi veľkosťou optického toku a miestom, kde je objekt vo vzťahu k nám. Ak zdvojnásobíme rýchlosť, ktorou cestujeme, optický tok, ktorý vidíme sa taktiež zdvojnásobí. Ak je objekt dvakrát bližšie k nám, optický tok bude opäť dvojnásobný. Predpokladáme, že cestujeme smerom vpred. Optický tok je najrýchlejší keď máme objekt po našom boku pod uhlom 90° alebo priamo nad nami alebo pod nami. Ak sa objekt približuje smerom vpred alebo smerom vzad, optický tok bude pomalší. Objekt priamo pred nami nebude mať žiadny optický tok a bude to vyzerať akoby stál v pokoji. Avšak, pretože hrany tohto objektu nie sú priamo pred nami, tieto hrany budú vyzerať akoby sa pohybovali a objekt sa bude javiť ako keby sa zväčšoval. Príklad optického toku môžeme vidieť na obrázku 3.



Obrázok 3: Príklad optického toku pri pohľade z lietadla [8]

Každému bodu obrazu optického toku zodpovedá dvojrozmerný vektor rýchlosti, ktorý opisuje smer a veľkosť rýchlosti pohybu v danom mieste obrazu. Túto metódu môžeme použiť v prípadoch kedy je statická kamera a pohybujúca sa scéna, pohybujúca sa kamera a statická scéna, alebo pohybujúca sa kamera aj scéna. Je známych viacero metód hľadania optického toku. Vo všeobecnosti ich môžeme rozdeliť na dva typy:

- hustý optický tok – optický tok sa počíta pre všetky body v obraze. Metódy sú veľmi časovo aj výpočtovo náročné. Medzi možné chyby patrí, že v obraze sa môžu vyskytovať plochy s rovnakým jasom, napríklad biely papier, či stena. Pre tieto body je veľmi ťažké nájsť správny pohyb. Najvýznamnejším predstaviteľom tohto typu je metóda *Horn-Schunck* [5]. Do tejto kategórie batrí aj metóda *Block matching* rozobraná v kapitole 4.3.

- riedky optický tok - optický tok sa počíta len pre predom definované body, ktoré by mali byť ľahko sledovateľné. Príkladom takýchto bodov môžu byť rohy alebo hrany. Časová náročnosť výpočtu riedkeho optického toku je podstatne menšia ako u hustého optického toku. Predstaviteľom tohto typu je metóda *Lucas-Kanade* [6].

3.2.1 Výpočet optického toku

Nech je X bod scény, ktorý má v čase t súradnice (x, y) . V čase $t + dt$ sa tento bod zobrazí do bodu $(x + dx, y + dy)$. Preto tento bod v čase $t + dt$ nadobúda rovnakú hodnotu jasu ako bol jas v bode (x, y) v čase t.

[1] Budeme uvažovať spojitý obraz. Označíme jas v mieste (x, y) obrazu v čase t ako $f(x, y, t)$. Po rozvoji do Taylorovej rady a zanedbaní členov vyšších radov platí:

$$f(x + dx, y + dy, t + dt) = \tag{3.2}$$

$$= f(x, y, t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt = \tag{3.3}$$

$$= f(x, y, t) + f_x dx + f_y dy + f_t dt. \tag{3.4}$$

Za predpokladu nemenného osvetlenia pri translačnom pohybe daným hodnotami dx, dy, platí:

$$f\left(x + dx, y + dy, t + dt\right) = f\left(x, y, t\right). \tag{3.5}$$

Z toho plynie, že

$$-f_t = f_x \frac{dx}{dt} + f_y \frac{dy}{dt}. \tag{3.6}$$

Cieľom výpočtu je určiť rýchlosť $r = (dx/dt, dy/dt)^{\mathrm{T}} = (u, v)^T$. Pre odhad rýchlosti platí:

$$-f_t = f_x u + f_y v = \nabla f r, \tag{3.7}$$

kde ∇ je dvojrozmerný gradient obrazu.

4 Metódy určenia optického toku

4.1 Lucas-Kanade

Tento algoritmus bol predstavený v roku 1981 a patrí azda k tým najpoužívanejším. Pri výpočte používa iba malé okolie patriace významným bodom. Problémom tohto algoritmu je, že je možné počítať iba malú zmenu optického toku, a to len do veľkosti integračného okna. Tento problém bol vyriešený pyramídovou implementáciou, ktorá umožnuje sledovať i väčšie pohyby ako je integračné okno.

Algoritmus využíva tri predpoklady:

- konštantný jas,
- malé pohyby,
- priestorová súdržnosť.

Konštantný jas znamená, že sledované body nemenia v čase svoj jas. Čo možeme vyjadriť:

$$I\left(x(t), y(t), t\right) = I\left(x(t+dt), y(t+dt), t+dt\right). \tag{4.1}$$

Druhý predpoklad znamená, že medzi snímkami dochádza len k malým pohybom sledovaných bodov. Inak povedané časové prírastky sú pomerne malé, vzhľadom na rozsah pohybu. Na vysvetlenie uvažujme jednorozmernú dimenziu. Úpravou rovnice konštantného jasu pre jeden rozmer dostaneme

$$\underbrace{\frac{\partial I}{\partial x}}_{I_x}\underbrace{\left(\frac{\partial x}{\partial t}\right)}_{v} + \underbrace{\frac{\partial I}{\partial t}}_{I_t}, \tag{4.2}$$

$$I_x v + I_t = 0 \Rightarrow v = -\frac{I_t}{I_x}, \tag{4.3}$$

kde I_x je priestorová derivácia jasu, I_t je časová derivácia jasu a v je rýchlosť pohybu. Teraz je potrebné nájsť pre danú funkciu rýchlosť pohybu v. K výpočtu použijeme iteratívny výpočet pomocou Newtonovej metódy. Kedže sa priestorová derivácia v jednej snímke nemení, stačí ju vypočítať len v prvej iterácií a v dalších počítať len časovú deriváciu.

Pre riešenie v 2D obraze nám v rovnici pribudne súradnica y, rovnica teda bude mať nasledujúci tvar

$$I_x u + I_y v + I_t = 0. \tag{4.4}$$

Tým však dostaneme jednu rovnicu a dve neznáme. Tento problém vyrieši posledný predpoklad, a to priestorová súdržnosť. To znamená, že predpokladáme, že všetky body ležiace v jednej rovine vykonávajú rovnaký pohyb. Využitím tohto predpokladu dostaneme sústavu rovníc

$$\underbrace{\begin{bmatrix} I_x(p1) & I_y(p1) \\ I_x(p2) & I_y(p2) \\ . & . \\ . & . \\ . & . \\ I_x(pn) & I_y(pn) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d} = - \underbrace{\begin{bmatrix} I_t(p1) \\ I_t(p2) \\ . \\ . \\ . \\ I_x(pn) \end{bmatrix}}_{b} \tag{4.5}$$

,

z ktorej vypočítame požadovanú rýchlosť pohybu. Túto sústavu je možné riešiť metódou najmenších štvorcov, ktorú môžeme zapísať:

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_y I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b}, \tag{4.6}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b. \tag{4.7}$$

Riešenie je možné určiť v prípade, že matica $A^T A$ je regulárna.

4.2 Horn-Schunck

Metóda Horn-Schunck vychádza taktiež z predpokladu priestorovej súdržnosti. Miera s akou sa optický tok môže líšiť od tohto predpokladu môže byť pre spojitý obraz v intervaloch $x\epsilon\,(0, M)\,,\ y\epsilon(0, N)$ vyjadrená pomocou integrálu

$$S = \int_0^N \int_0^M \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 dxdy. \tag{4.8}$$

Ako vyplýva z rovnice 4.8, predpokladáme, že výsledok tejto rovnice bude nulový. Avšak kvôli šumu vyskytujúcemu sa v obraze výsledok nulový nebude. Mieru tejto odchýlky môžeme vypočítať integrálom na rovnakých spojitých intervaloch

$$C = \int_0^N \int_0^M \left(\frac{\partial E}{\partial x} u + \frac{\partial E}{\partial y} v + \frac{\partial E}{\partial t} \right)^2 dxdy. \tag{4.9}$$

Kvôli týmto odchylkám je dobré si zaviesť konštantu α, ktorá vyjadruje úmernosť chybovej hodnoty C k šumu v obraze. Celková miera odlišnosti od vyššie uvedeného predpokladu je

$$f = S + \alpha C. \tag{4.10}$$

Po minimalizácií tejto funkcie dostaneme dve rovnice s dvoma neznámymi u a v

$$E^2 u + E_x E_y v = \alpha \nabla^2 u - E_x E_t, \tag{4.11}$$

$$E^2 v + E_x E_y u = \alpha \nabla^2 u - E_y E_t.$$

Pretože obraz zaznamenaný videokamerou je spojitý, môžeme použiť Laplacián:

$\nabla^2 u = (\overline{u} - u)$,

$\nabla^2 v = (\overline{v} - v)$,

kde $\overline{u}$ a $\overline{v}$ sú lokálne priemery dané konvolučným jadrom:

$$\begin{pmatrix} \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \\ \frac{1}{6} & -1 & \frac{1}{6} \\ \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \end{pmatrix}.$$

S použitím tejto aproximácie je možné rovnice 4.11 zapísať

$$(\alpha + E_x^2) u + E_x E_y v = (\alpha \overline{u} - E_x E_t), \tag{4.12}$$

$$(\alpha + E^2 yv) + E_x E_y u = (\alpha \overline{v} - E_y E_t).$$

Riešením pre u a v je

$$u = \frac{\left(\alpha + E_y^2\right) \overline{u} - E_x E_y \overline{v} - E_x E_t}{(\alpha + E_x^2 + E_y^2)}, \tag{4.13}$$

$$v = \frac{\left(\alpha + E_y^2\right) \overline{v} - E_x E_y \overline{u} - E_y E_t}{(\alpha + E_x^2 + E_y^2)}.$$

Pretože by bol výpočet pre každý pixel veľmi náročný, je vhodné použít' niektorú z iteračných metód. Napríklad Gauss-Seidelovú metódu:

$$u^{n+1} = \overline{u}^n - E_x \frac{E_x \overline{u}^n + E_y \overline{v}^n + E_t}{(\alpha + E_x^2 + E_y^2)}, \tag{4.14}$$

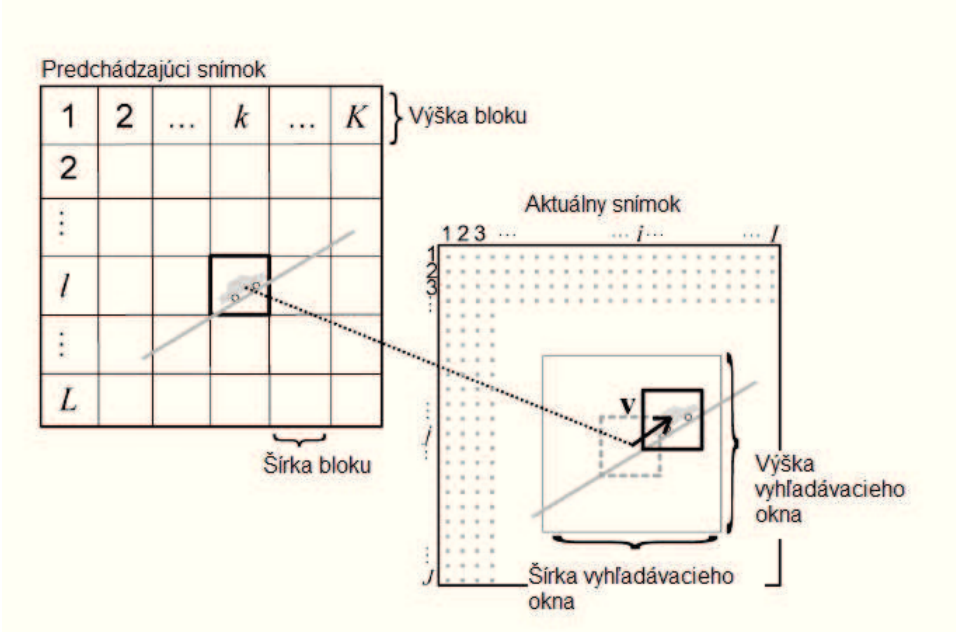
$$v^{n+1} = \overline{v}^n - E_y \frac{E_x \overline{u}^n + E_y \overline{v}^n + E_t}{(\alpha + E_x^2 + E_y^2)}.$$

Hodnota zložiek vektoru rýchlosti je výsledkom priemernej hodnoty rýchlosti danej zložky v danom okolí určené predchádzajúcou iteráciou a hodnota príslušnej smerovej zložky gradientu jasu v danom bode, vypočítaná na základe hodnot optického toku z predchádzajúcej iterácie a gradientu jasu v odpovedajúcom bode.

4.3 Block matching

V testovacej aplikácií bola implementovaná metóda pre výpočet optického toku. Jedná sa o metódu nazývanú block matching, ktorá patrí do skupiny hustého optického toku, teda počíta optický tok pre každý bod obrázku. Túto metóda je používaná napríklad v štandardoch pre kompresiu videa H.26X a MPEG.

Podstatou tohto algoritmu je rozdeliť prvý obrázok zo sekvencie na neprekrývajúce sa časti, nazývané bloky a nájsť zodpovedajúci blok vo vyhľadávacej oblasti druhého obrázku. Tým zistíme vektor pohybu bodov v bloku.



Obrázok 4: Block matching, predchádzajúci a súčastný snímok, vektor posunutia v

Ideálnym prípadom by bolo keby sa všetky body bloku zhodovali. To sa však v reálnej sekvencií obrázkov príliž často nevyskytuje, pretože pohybujúce sa objekty z pohľadu pozorovateľa menia svoj tvar, mení sa svetlo odrážajúce sa od objektov a taktiež sa v snímkoch vyskytuje šum. Navyše scény obsahujú príchody nových objektov ako aj miznutie niektorých objektov. Aj napriek týmto problémom vznikli mnohé techniky na meranie podobnosti blokov. Medzi najčastejšie používané kritéria patria:

- *mean absolute distance,*
- *mean squared distance,*

- *normalized cross-correlation,*
- *sum of absolute differences (4.15,)*
- *sum of squared differences.*

$$SAD = \sum |I(x, y, t) - I(x, y, t)| \tag{4.15}$$

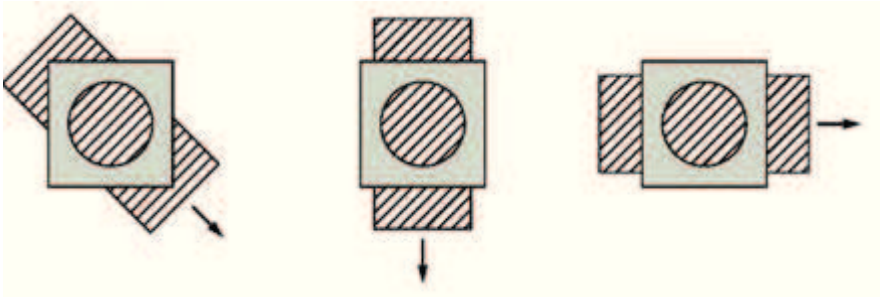
Pre nás najzaujímavejšou technikou je SAD, pre jej jednoduchosť a rýchlosť. Pracuje tak, že vypočíta absolútnu hodnotu rozdielu každého pixelu bloku prvého obrázka a bloku aktuálne porovnávaného bloku druhého obrázka. Tieto rozdiely sú sčítané a výsledok predstavuje podobnosť medzi blokmi. Čiže čím je hodnota SAD menšia, tým sú bloky podobnejšie.

4.3.1 Veľkosť bloku

Výber správnej veľkosti bloku nie je triviálna záležitosť. Vo všeobecnosti sú väčšie bloky menej citlivé na šum, avšak nemusia zachytiť malé pohyby, alebo pohyby malých objektov. Najdôležitejším faktorom pri výbere veľkosti bloku je veľkosť objektov, ktoré majú byť sledované. Ďalšími činiteľmi sú množstvo šumu v snímkach, textúry objektov a pozadia. Štruktúra objektov vedie k tzv. *Aperture problému*.

Ako je vidieť na obrázku 5 *Aperture problém* spočíva v tom, že rozličné fyzikálne pohyby sú v niektorých prípadoch nerozpoznateľné. Z časti je možné sa tomuto problému vyhnúť väčšou veľkosťou bloku.

Pri kompresii videa sa väčšinou používajú bloky veľkosti 8x8 alebo 16x16 pixelov.



Obrázok 5: Aperture problem

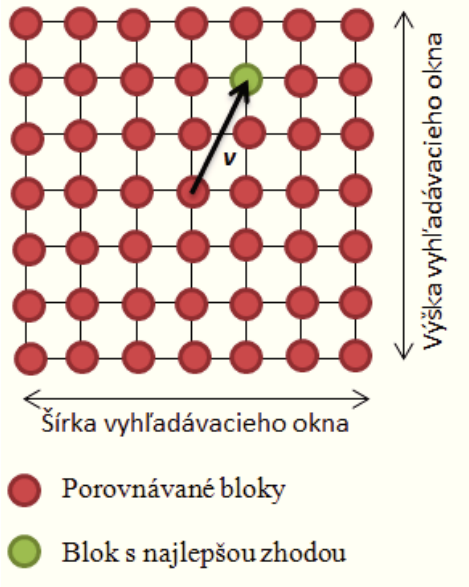
4.3.2 Vyhľadávacia oblasť

Veľkosť vyhľadávacieho okna je dôležitá pre nájdenie správnej zhody. S veľkosťou vyhľadávacieho okna rastie aj výpočetná záťaž(druhá mocnina veľkosti okna). Existujú aj niektoré suboptimálne algoritmy, ktoré vyhľadávajú len v predpovedanom smere pohybu, vtedy je výpočetná záťaž menšia.

4.3.3 Vyhľadávacie algoritmy

4.3.4 Full Search

Pre porovnávanie blokov boli vyvinuté rôzne druhy algoritmov. Jeden z prvých používaných algoritmov bol *Full Seacrh* alebo *Exhaustive Search*. Podstatou tohto vyhľadávania je každý blok prvého obrázka porovnať s každým blokom vyhľadávacieho okna druhého obrázka a tak nájsť najlepšiu zhodu.



Obrázok 6: Príklad algoritmu *Full Search*

Výhodou *Full Seacrh* algoritmu je 100 % istota, že nájdeme absolútne minimum vo vyhľadávacom okne.

Naopak nevýhodou je vysoká zložitosť algoritmu, ktorá je priamo úmerná druhej mocnine rozsahu hľadania. Z toho vyplýva, že veľkosť vyhľadávacieho okna je veľmi dôležitá pre rýchlosť vyhľadávania.

4.3.5 Fast-search algoritmy

Neskôr bolo vyvinutých mnoho ďalších algoritmov, ktoré sa stali populárne vďaka nízkej zložitosti a vysokej rýchlosti. Ich rýchlosť spočíva hlavne v malom množstve porovnávaných blokov. Štúdiu týchto algoritmov sa venuje [4].

My si popíšeme tieto tri:

- Three-Step Search,
- Two Dimensional Logarithmic Search,
- Hierarchial Search.

Three-Step Search

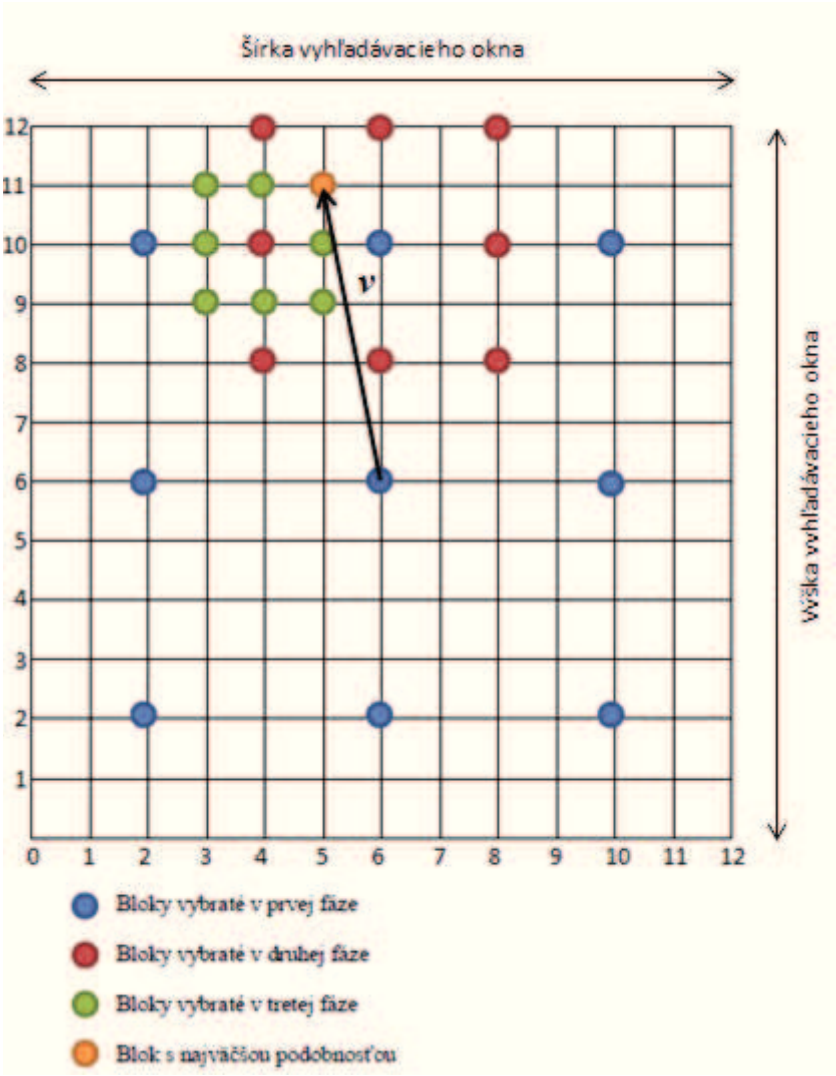
Tento algoritmus uviedol Koga et al. v roku 1981 a implementoval Lee et al . Patrí medzi suboptimálne algoritmy, pretože jeho správnosť je podmienená štruktúrou obrázku.

Algoritmus môžeme popísať nasledovne:

1. krok Three-Step Search začína vypočítaním stredu vyhľadávacej oblasti. Štartovacia veľkosť kroku je nastavená na polovicu veľkosti vyhľadávacej oblasti. Vyhľadává sa na ôsmych pozíciách vzdialených veľkosťou kroku od stredu. Vypočítajú sa hodnoty SAD blokov na jednotlivých bodoch, tieto hodnoty sa porovnajú a vyberie sa blok s najväčšou podobnosťou. Na obrázku 7 bol v tomto kroku vybratý blok na pozícií (10,6).

2. krok Veľkosť kroku sa podelí dvoma, za stred sa určí najlepšia zhoda z 1. kroku. Znovu sa sa hľadá najmänšia hodnota SAD na deviatich pozíciách. Na obrázku 7 bol v tomto kroku vybratý blok na pozícií (10,4).

3. krok Celý proces sa opakuje až pokiaľ sa veľkosť kroku nerovná 1. Na obrázku 7 bol v poslednom kroku vybratý blok na pozícií (11,5), ktorý predstavuje najlepšiu zhodu a môžeme z neho získať vektor pohybu.



Obrázok 7: Príklad algoritmu *Three-Step Search*

Z obrázka 7 je jasné že algoritmus je oveľa rýchlejší ako *Full Search*. Pri veľkosti vyhľadávacieho okna 13x13 pixelov *Full Search* porovná 169 blokov, zatiaľ čo *Three-Step Search* len 25 blokov.

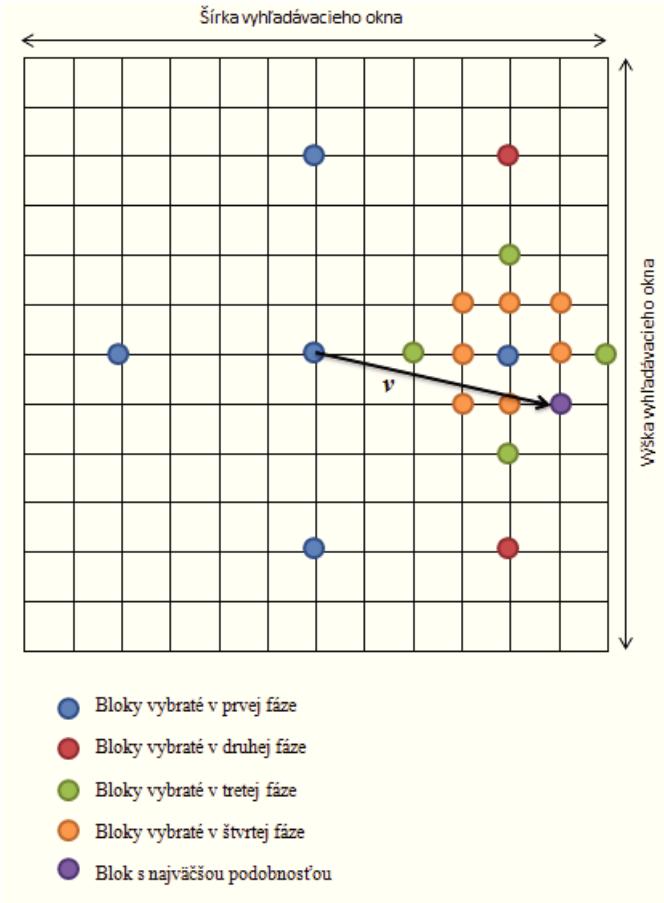
Two Dimensional Logarithmic Search

Algoritmus uviedol v roku 1981 Jain Jain. Oproti *Three-Step Search* vyžaduje viac krokov, avšak môže byť presnejší, zvlášť v prípade rozmerného vyhľadávacieho okna. Algoritmus popisujú nasledujúce kroky:

1. krok Na začiatku sa určí veľkosť kroku. Porovnajú sa štyri bloky umiestnené na osy X a Y vo vzdialenosti veľkosti kroku.

2. krok Ak je pozícia najlepšej zhody v strede, krok sa podelí dvoma. Inak sa najlepšia zhoda stane stredom a opakuje sa 1. krok.

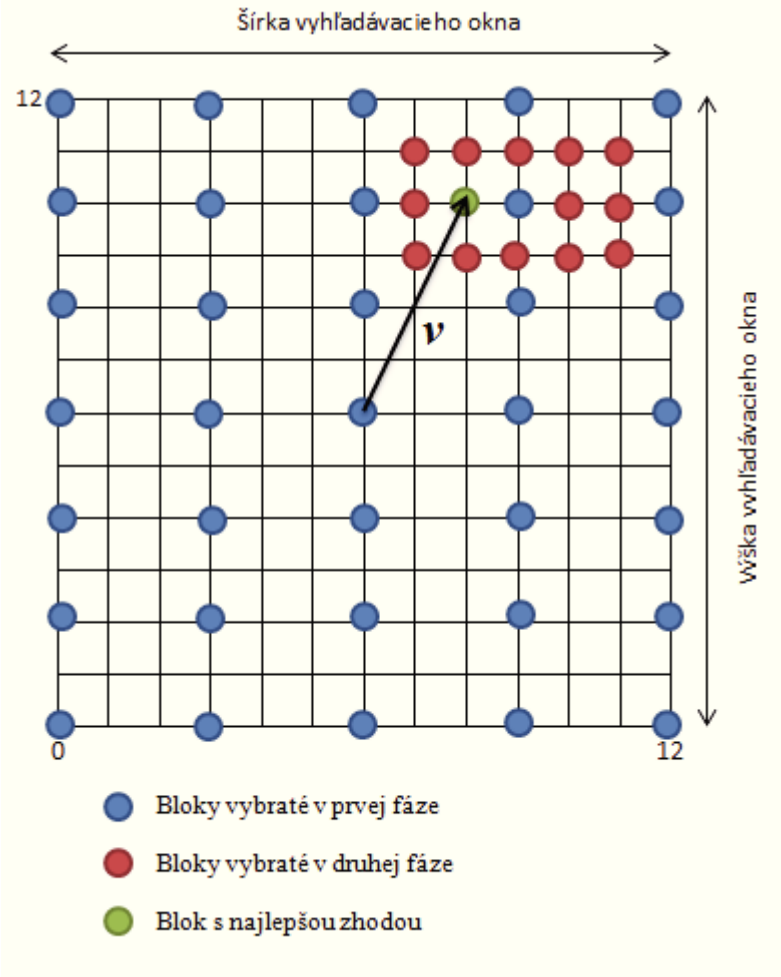
3. krok Keď veľkosť kroku nadobudne hodnotu jedna, okolo stredu sa vytvorí 9 blokov z ktorých je vybratá najlepšia zhoda.



Obrázok 8: Príklad algoritmu *Two Dimensional Logarithmic Search*

Hierarchial search

Jedna z najjednoduchších aplikácií je dvoj-úrovňový *Hierarchial search* [10]. V prvom kroku tohto algoritmu sa vytvorí riedka sieť blokov. Vyberie sa sa blok s najlepšou zhodou, ktorý sa stane stredovým bodom pre jemnejšiu sieť blokov. Z nej sa nakoniec vyberie blok s najlepšou zhodou, ktorý určuje vektor posunutia. Z príkladu na obrázku 9 je jasné ako fungujú tieto algortmy pri viacerých úrovňach. Podľa [4] je *Hierarchial search* je jeden z *Fast-search* algoritmov, ktorý sa svojou kvalitou najviac približuje *Full Search* vyhľadávaniu a pritom výrazne znížuje výpočetnú zátaž.



Obrázok 9: Príklad algoritmu *Hierarchial search*

5 GPGPU

Procesor grafickej karty (GPU) je príkladom mnohojadrovej technológie, ktoré zaznamenávajú v posledných rokoch vzrastajúci trend. GPU boli pôvodne vyvinuté pre priemysel počítačových hier, aby umožnili real-time 3D grafiku s vysokým rozlišením, avšak ukázalo sa, že sú užitočné aj pre množstvo negrafických aplikácií. Používanie GPU pre negrafické aplikácie sa označuje GPGPU (General-purpose computing on graphics processing units).

Nakoľko je GPU vysoko paralelné SIMD zariadenie, algoritmy optického toku s vysokou úrovňou paralelizmu môžu dosiahnut' na grafickom hardvéry lepší výkon ako iteratívne algoritmy.

5.1 CUDA

Compute Unified Device Architecture (CUDA) od spoločnosti nVidia je najstaršou zverejnenou technológiou pre paralelné výpočty na GPU. Architektúra je dostupná iba na vybraných akcelerátoroch spoločnosti nVidia. Táto technológia rozširuje programovací jazyk C tým, že umožnuje programátorovi definovat'C funkcie, ktoré sú vykonávané paralelne veľkým počtom vlákien na kompatibilnom GPU hardvéri.

5.1.1 Kernely, bloky, vlákna

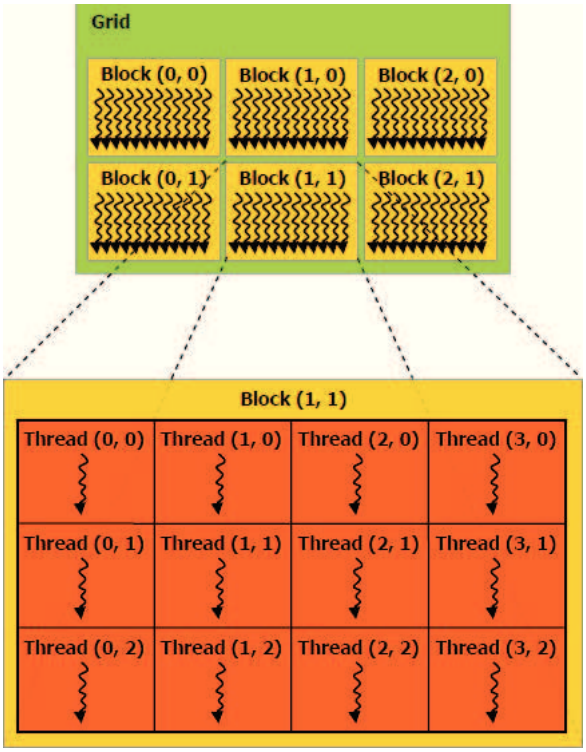
CUDA program je rozdelený na časti, ktoré sa vykonávajú na CPU a na časti, ktoré sa vykonávajú na GPU. Sekcie, ktoré bežia na GPU sa volajú *kernely*. Sú to funkcie, ktoré sú vykonávané každým spusteným vláknom. Označujú sa kľúčovým slovom *__global__*.

Vlákna sú organizované do jedna, dvoj alebo trojrozmerných *blokov*. Vlákna v rovnakom bloku môžu zdielat' data a môžeme synchronizovat' ich beh. Počet vlákien v bloku je závislý na konkrétnom zariadení. Vlákna musia byt' nezávislé, pretože nie je zaručené v akom poradí budú spustené. Každé vlákno má v rámci kernelu svoje identifikačné číslo, programátorovi prístupné cez preddefinovanú premennú *threadIdx*. *ThreadIdx* je typu dim3, čo je trojzložkový (x,y,z) vektor. Trojzložkový vektor kvôli uľahčeniu práce s maticami, vektormi a viacrozmernými poliami.

Bloky sú organizované do jedna, dvoj alebo trojrozmernej mriežky. Každý blok je v v rámci mriežky identifikovateľný pomocou premennej *blockIdx*. Bloky vlákien pracujú nezávisle na ostatných.

Vlákna spracovávané v jednom okamihu sa označujú *warp*. Veľkosť warp-u je závislá na počte výpočetných jednotiek.

Výber usporiadania mriežok a blokov je v CUDA programe ponechaný na programátorovi a učuje sa pri spúštaní kernelu [3].

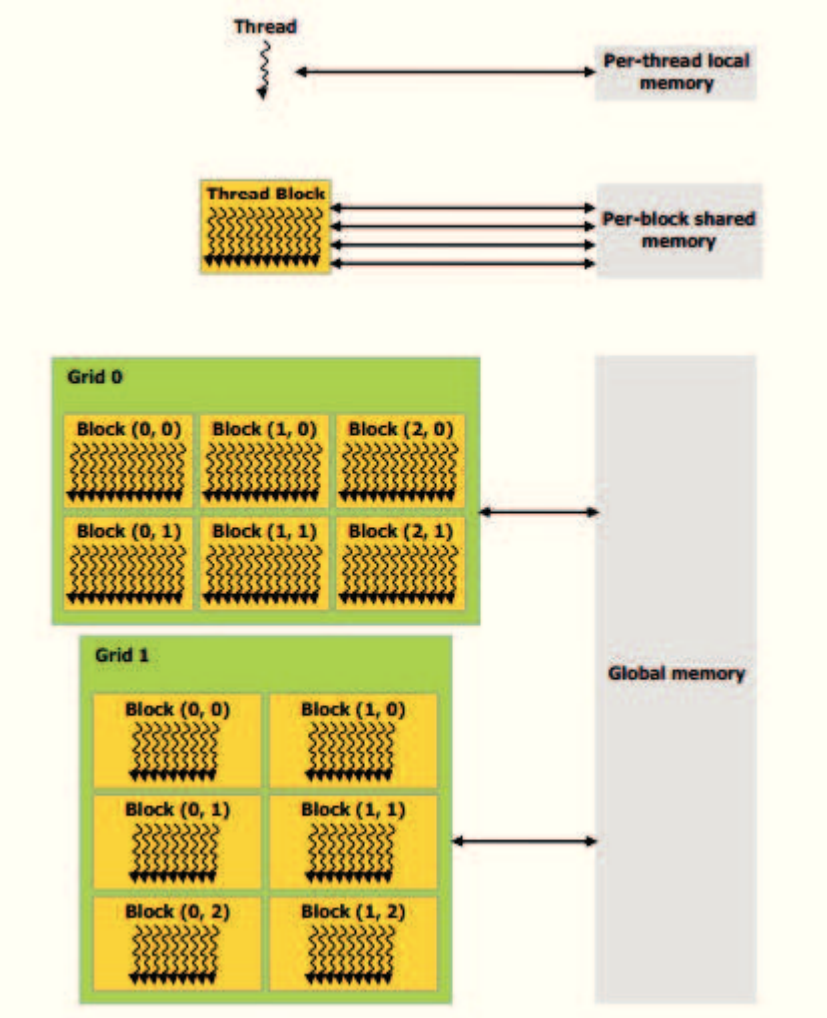


Obrázok 10: Vlákna sú zoskupené do blokov a bloky do mriežok.

Ako je znázornené na obrázku 11 každé CUDA vlákno môže pristupovať k:

- poli registrov - každé vlákno môže pristupovať len k svojím registrom
- lokálnej pamäti - používa sa v prípade, že dojde k vyčerpaniu registrov. Táto pamäť je prístupná len jednomu vláknu, ale fyzicky je umiestnená v globálnej pamäti akcelerátoru a preto je prístup k nej pomalejší ako napr. k zdieľanej pamäti.
- zdieľanej pamäti - kešovaná pamäť, ktorá je viditeľná všetkým vláknam v jednom bloku. Pristupuje sa k nej cez brány nazývané *banky*. Každý bank môže sprístupniť len jednu adresu v takte.

- globálnej pamäti - určená pre kopírovanie dat z RAM do GPU a naopak. Je zdieľaná medzi všetkými vláknami a nie je ukladaná do cache pamäti.
- pamäti pre konštanty - určená len pre čítanie, je zdieľaná medzi všetkými vláknami
- pamäti pre textúry - cachovaná a taktiež zdieľaná medzi všetkými vláknami.



Obrázok 11: Pamäťový model CUDA

Typický priebeh GPGPU výpočtu môžeme popísať nasledovne:

1. vyhradenie pamäti na GPU,
2. presun z pamäti RAM do pamäti grafického akcelerátoru,
3. spustenie výpočtu na grafickej karte,
4. presun výsledkov z pamäti grafickej karty do RAM pamäti.

Presun dát medzi RAM a pamäťou grafickej karty je veľmi pomalý preto je ideálne tieto presuny minimalizovať.

6 Implementácia

Jedným z hlavných cieľov bakalárskej práce bolo vybrať a naimplementovať jeden z algoritmov určenia optického toku. Po dohode s vedúcim bakalárskej práce bol vybraný algoritmus *Block matching* s vyhľadávacím algoritmom *Full Search*. Testovacia aplikácia je implementovaná v programovacom jazyku C/C++. Cieľom aplikácie je z dvoch snímkov sekvencie určiť optický tok.

Algoritmus je jednoduchý a priamočiary. Predchádzajúci snímok I rozdeliť na bloky s rozmermi m x n a pre každý blok bI vytvoriť na každom bode vyhľadávacej oblasti druhého snímku J blok bJ s rovnakými rozmermi a vypočítať hodnotu SAD

$$SAD_{bI,bJ} = \sum_{x,y=0}^{m,n} |I(x,y) - J(x,y)|. \tag{6.1}$$

Nakoniec vybrať z týchto hodnôt tú najmenšiu,

$$SAD_{min} = \min\{SAD_{bI_n,bJ_1}, SAD_{bI_n,bJ_2}, SAD_{bI_n,bJ_3}, ..., SAD_{bI_n,bJ_m}\},$$

kde m je počet porovnávaných blokov vo vyhľadávacej oblasti druhej snímky, a tým určiť vektor posunutia v tohto bloku

$$v = (bJ_x - bI_x, bJ_y - bI_y),$$

kde bI_x a bI_y sú súradnice hľadaného bloku a bJ_x a bJ_y sú súradnice bloku s najvačšou podobnosťou.

CUDA implementácia

V CUDA aplikácií budú dva kernely:

fullSearch kernel - každé vlákno bude počítať hodnotu SAD pre jeden blok z predchádzajúceho snímku a jeden blok z aktuálného snímku. Hodnotu SAD uloží do globálnej pamäti. Pre každý blok budeme mať toľko vlákien koľko má kandidátov.
Na základe *threadIdx, blockDim a blockIdx* si určí id bloku, podľa neho pozíciu bloku, a pozíciu kandidáta, pre ktoré sa bude hodnota SAD počítať. V jednoduchom cykle prejde všetky body blokov a spočíta túto hodnotu, uloží ju do globálnej pamäti spolu so súradnicami blokov.

findMinimum kernel - V tomto kerneli budeme hľadať minimálnu hodnotu SAD. Pre každý blok budeme mať jedno vlákno, ktoré bude v globálnej pamäti prehľadávať hodnoty SAD patriace tomuto bloku.

Počet blokov bude teda v prvom kerneli *(pB x pK)/pV* a v druhom kerneli *pB/pV*, kde pB je počet blokov, pK je počet blokov vo vyhľadávacej oblasti a pV je počet vlákien v bloku. Veľkosť bloku bude 16x16 vlákien.

6.1 Sekvencie obrázkov

Pre jednoduchosť aplikácie predávame snímky parametrom pri spustení, spolu s veľkosťou bloku a veľkosťou vyhľadávacej oblasti. Po načítaní snímkov sú obidva snímky prevedené na stupne šedej. Obidva obrázky sú, ako aj ostatné 2D dáta, reprezentované jednorozmerným poľom. Čiže bod obrázka na pozícií (x, y) bude mať index

$$index = y * width + x,$$

kde $width$ je šírka snímky.

6.2 Organizácia kódu

Aplikácia obsahuje CPU a zároveň i GPU implementáciu tohto algoritmu. CUDA kernely sú uložené vo zvlášť súbore s príponou *.cu. Zdrojový kód je rozdelený do nasledujúcich súborov:

- *FlowCPU.cpp* - implementácia pre CPU,
- *FlowGPU.cu* - kernely pre výpočet algoritmu na GPU,
- *main.cpp* - načítanie obrázkov, volanie funkcií pre výpočet na GPU a CPU, zobrazenie výsledku, porovnanie časov vykonávania.

6.3 Knižnica OpenCV

Pri implementovaní testovacej aplikácií je použitá knižnica OpenCV [9]. Je to open-source knižnica, ktorá je zameraná na real-time počítačové videnie. Má podporu pre C/C++ a Python a beží na operačných systémoch Windows, Linux, Android a MacOS. Obsahuje viac ako 2500 optimalizovaných algoritmov.

V aplikácií je knižnica použitá pri načítaní obrázkov, pri prevádzaní na stupne šedej a pri zobrazovaní výsledku.

7 Testovanie

Na testovacie účely bol poskytnutý počítač Tesla01, ktorý je určený pre vývoj paralelných aplikacií na grafických procesorových jednotkách nVidia v prostredí CUDA. Na tento počítač sa pristupovalo pomocou vzdialenej plochy.

Operačný systém	Linux CentOS
CPU	2x Dual-Core AMD Opteron 2218
GPU	2x NVIDIA TESLA C2050 (2 x 448 jadier)

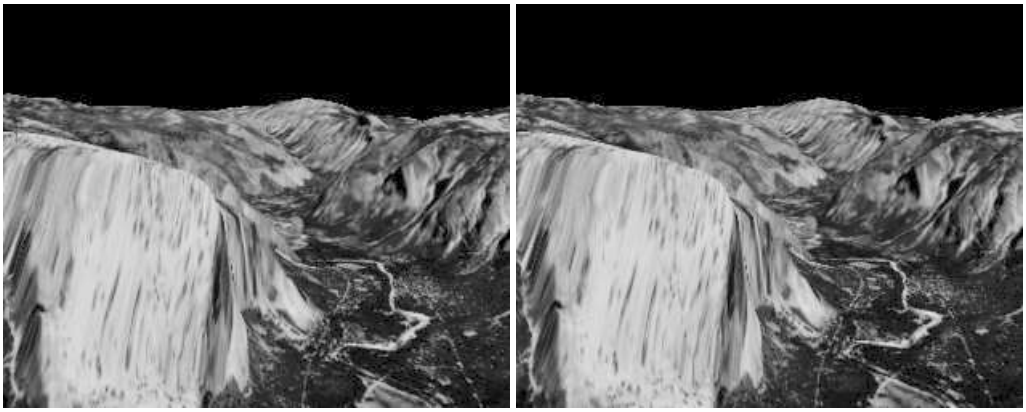
Tabuľka 1: Testovací stroj Tesla01

Je zrejmé, že výsledný čas vykonávania algoritmu bude závisieť od veľkosti snímkov, veľkosti vyhľadávacej oblasti a od rozmerov bloku. Preto sme vykonali sériu testov, v ktorých sme tieto parametre menili a merali výsledný čas a následne porovnávali časy vykonávania.

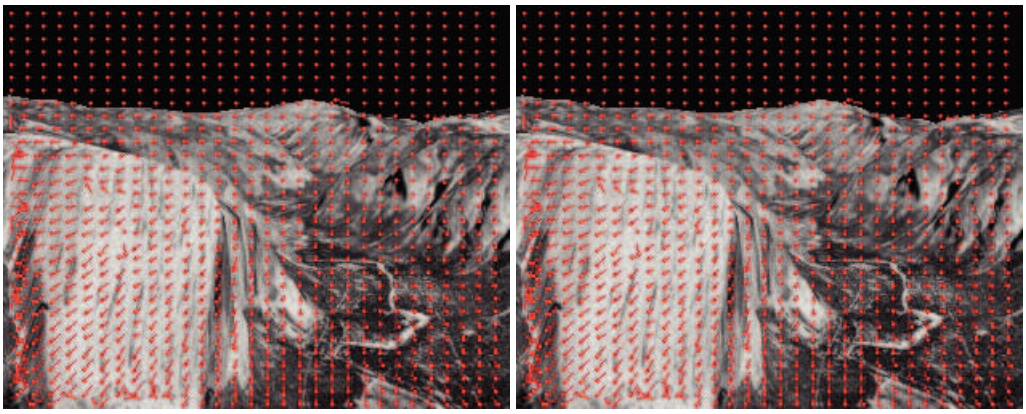
Boli testované štyri rôzne sekvencie snímkov s rozlišením 316x252 pixelov, 720x480 pixelov, 1280x720 pixelov a 1600x1200 pixelov. Každý snímok bol rozdelený na 20 a 30 blokov a veľkosť vyhľadávacej oblasti bola vždy nastavená na dvojnásobok veľkosti bloku. Celkovo sme teda vykonali 8 testov. Po každom teste sme vizuálne skontrolovali správnosť výsledku a odčítali čas spracovávania na CPU a GPU a tým dostali pomer zrýchlenia.

V prvom meraní bola teda testovaná sekvencia s rozlišením 316x252 pixelov (obrázok 12), veľkosťou bloku 15x12 pixelov a veľkosťou vyhľadávacej oblasti 30x24 pixelov. Ako vidíme na obrázku 13, výsledné vektory sú rovnaké. Výsledný čas na GPU je 279ms a na CPU je 1198ms. Čiže výpočet na GPU bol približne 4 krát rýchlejší.

Pri veľkosti bloku 10x8 pixelov a vyhľadávacej oblasti 20x16 pixelov bol výsledný čas na GPU 6 krát kratší.



Obrázok 12: Vstupná sekvencia



Obrázok 13: Výsledné vektory pohybu blokov, vľavo CPU, vpravo GPU

Pri sekvencií s rozlišením 720x480 pixelov, ktorý vidíme na obrázku 14, a bloku s veľkosťou 14, už môžeme vidiet'25-násobné urýchlenie. Výsledne vektory sú na obrázku 15

Ako môžeme vidiet', tak už pri neoptimalizovanom programe sú urýchlenia celkom slušné. Jednou z možností ako tento kód optimalizovat' a teda aj urýchlit', je použitie textúrovacej pamäte. V tomto prípade boli obidva snímky nahraté do textúrovacej pamäti a testy sme zopakovali.

Obrázok 14: Vstupná sekvencia

Obrázok 15: Výsledné vektory pohybu blokov

Pri použití tejto pamäte bolo urýchlenie oveľa väčšie. Pri testovaní sekvencie s rozlíšením 1600x1200 pixelov veľkosti bloku 53x40 a vyhľadávacej oblasti s rozmermy 106x80 bol výpočet na GPU až 198 krát rýchlejší. V tabuľke 2 môžeme vidieť všetky namerané hodnoty.

Obrázok	Blok	Vyhľadávacia oblasť	Čas CPU[ms]	Čas GPU[ms]	**Urýchlenie**
316x252	15x12	30x24	1524	140	**10**
720x480	36x24	72x48	17198	220	**78**
1366x720	68x36	136x72	118963	637	**186**
1600x1200	80x60	160x120	488683	2526	**193**
316x252	10x8	20x16	1009	134	**7**
720x480	24x16	48x32	7865	179	**43**
1366x720	45x24	90x48	54657	390	**140**
1600x1200	53x40	106x80	225322	1134	**198**

Tabuľka 2: Porovnanie rýchlosti algoritmu na CPU a na GPU.

8 Záver

Ciele bakalárskej práce vytýčené v úvode práce sa podarilo splniť. Práca oboznamuje o metódach určenia optického toku, o jeho algoritmoch a aplikáciach. Jednou z hlavných úloh práce bolo tiež naimplementovať jeden z algoritmov určenia optického toku v jazyku C/C++ a zároveň pre prostredie CUDA. Následne otestovať tento algoritmus a porovnať čas vykonávania na CPU a GPU.

Pri testoch boli použité rôzne veľkosti obrázkov, rozmery blokov a vyhľadávacích okien. Vizuálne boli porovnané výsledné optické toky vypočítané na CPU a na GPU a zároveň bol porovnaný čas vykonávania týchto algoritmov.

Vzhľadom k tomu že sa nám podarilo technológiou CUDA urýchliť algoritmus výpočtu optického toku v niektorých prípadoch až skoro 200 násobne, môžeme túto platformu určiť ako vhodnú pre výpočtovo náročné úlohy spojené s analýzou pohybu v obraze.

Algoritmy, ktoré sú popísané v práci ponúkajú mnoho možností na vylepšenia alebo rozšírenia. Jednou z možností by mohlo byť predspracovanie sekvencií obrazkov vo forme odstránenia šumu. Týmto vylepšením by sa určite zlepšila kvalita algoritmu. Určite by bolo zaujimavé implementovať všetky popísané algoritmy v prostredí CUDA a porovnať ich urýchlenie.

9 Reference

[1] Hlaváč, Šonka, *Počítačové vidění,* Praha: GRADA, 1992.

[2] A. Gyaourova, C. Kamath, S.-C. Cheung *Block Matching for Object Tracking*, 2003

[3] NVIDIA Corporation *NVIDIA CUDA C Programming Guide 4.1*, 2011
Dostupné z: <http://developer.nvidia.com>

[4] Deepak Turaga, Mohamed Alkanhal *Search Algorithms for Block-Matching in Motion Estimation*, 1998

[5] Berthold K.P. Horn, Brian G. Rhunck *Determining Optical Flow*
Dostupné z: <http://www.csd.uwo.ca/faculty/beau/CS9645/PAPERS/Horn-Schunck.pdf>

[6] Bruce D. Lucas, Takeo Kanade *An Iterative Image Registration Technique with an Application to Stereo Vision*
Dostupné z: <http://goo.gl/n24GI>

[7] *Optical Flow — Centeye, Inc.* [online]. 2003 [cit. 2012-04-03].
Dostupné z: <http://centeye.com/technology/optical-flow/>

[8] Visual Perception Theory. *Simply Psychology* [online]. 2007 [cit. 2012-04-24].
Dostupné z: <http://www.simplypsychology.org/perception-theories.html>

[9] *OpenCV* [online]. 2010 [cit. 2012-04-20].
Dostupné z: <http://opencv.willowgarage.com>

[10] Colin E. Manning *Video Compression Explained* [online]. 1996 [cit. 2012-03-28]
Dostupné z: <http://www.newmediarepublic.com/dvideo/>

A Obsah priloženého CD

Obsah priloženého CD:

- Testovacia aplikácia optFlow
- Zložka s kompletnými zdrojovými kódmy - `/src/`
- Zložka so sadou testovacích obrázkov - `/testingImages/`
- Návod na použitie aplikácie - `README.TXT`
- Textová časť bakalárskej práce v elektronickej podobe - `BP_suc0026.pdf`